Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота № 6**

**з курсу: «Теорія проектування**

**комп’ютерних систем»**

*Виконав:*

студент 4 курсу

групи ІО-12, ФІОТ

Бута С. О.

Київ, 2014 р.

**ЗАДАНИЕ:**

1. Разработать процедуру минимизации булевых функций методов Квайна-МакКласки

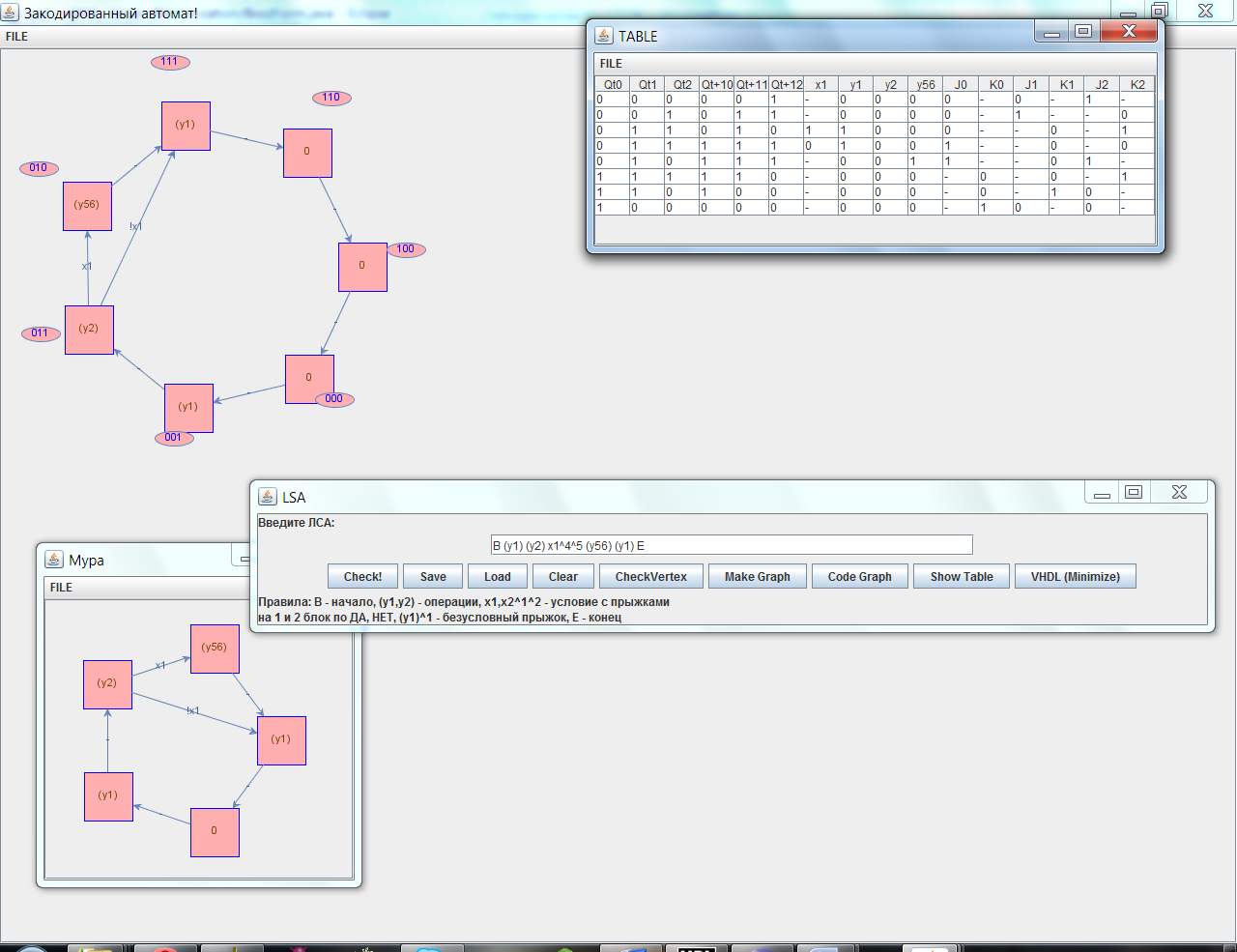
2. Разработать средства анализа эффективности минимизации по количеству элементов и количеству входов и выходов.

3. На основании разработанной процедуры в пункте 1, реализовать модуль минимизации булевых функций переходов и функций возбуждения триггеров из таблицы, построенной в предыдущей работе. Отобразить неминимизированные/минимизированные функции и эффективность их минимизации.

4. Реализовать средства сохранения результатов минимизации в файл.

**1205 = 10000010000011**

**Программа:**



Функции переключения и выходных сигналов:

y1 = !(!Qt0 !Qt1 !Qt2 v !Qt0 !Qt1 Qt2 v !Qt0 Qt1 !Qt2 v Qt0 Qt1 Qt2 v Qt0 Qt1 !Qt2 v Qt0 !Qt1 !Qt2 )

y2 = !(!Qt0 !Qt1 !Qt2 v !Qt0 !Qt1 Qt2 v !Qt0 Qt1 Qt2 x1 v !Qt0 Qt1 Qt2 !x1 v !Qt0 Qt1 !Qt2 v Qt0 Qt1 Qt2 v Qt0 Qt1 !Qt2 v Qt0 !Qt1 !Qt2 )

y56 = !(!Qt0 !Qt1 !Qt2 v !Qt0 !Qt1 Qt2 v !Qt0 Qt1 Qt2 x1 v !Qt0 Qt1 Qt2 !x1 v Qt0 Qt1 Qt2 v Qt0 Qt1 !Qt2 v Qt0 !Qt1 !Qt2 )

J0 = !(!Qt0 !Qt1 !Qt2 v !Qt0 !Qt1 Qt2 v !Qt0 Qt1 Qt2 x1 v Qt0 Qt1 Qt2 v Qt0 Qt1 !Qt2 v Qt0 !Qt1 !Qt2 )

K0 = !(!Qt0 !Qt1 !Qt2 v !Qt0 !Qt1 Qt2 v !Qt0 Qt1 Qt2 x1 v !Qt0 Qt1 Qt2 !x1 v !Qt0 Qt1 !Qt2 v Qt0 Qt1 Qt2 v Qt0 Qt1 !Qt2 )

J1 = !(!Qt0 !Qt1 !Qt2 v !Qt0 Qt1 Qt2 x1 v !Qt0 Qt1 Qt2 !x1 v !Qt0 Qt1 !Qt2 v Qt0 Qt1 Qt2 v Qt0 Qt1 !Qt2 v Qt0 !Qt1 !Qt2 )

K1 = !(!Qt0 !Qt1 !Qt2 v !Qt0 !Qt1 Qt2 v !Qt0 Qt1 Qt2 x1 v !Qt0 Qt1 Qt2 !x1 v !Qt0 Qt1 !Qt2 v Qt0 Qt1 Qt2 v Qt0 !Qt1 !Qt2 )

J2 = !(!Qt0 !Qt1 Qt2 v !Qt0 Qt1 Qt2 x1 v !Qt0 Qt1 Qt2 !x1 v Qt0 Qt1 Qt2 v Qt0 Qt1 !Qt2 v Qt0 !Qt1 !Qt2 )

K2 = !(!Qt0 !Qt1 !Qt2 v !Qt0 !Qt1 Qt2 v !Qt0 Qt1 Qt2 !x1 v !Qt0 Qt1 !Qt2 v Qt0 Qt1 !Qt2 v Qt0 !Qt1 !Qt2 )

(тоже что и выше, но в битах):

y1 = !(000-v001-v010-v111-v110-v100-)

y2 = !(000-v001-v0111v0110v010-v111-v110-v100-)

y56 = !(000-v001-v0111v0110v111-v110-v100-)

J0 = !(000-v001-v0111v111-v110-v100-)

K0 = !(000-v001-v0111v0110v010-v111-v110-)

J1 = !(000-v0111v0110v010-v111-v110-v100-)

K1 = !(000-v001-v0111v0110v010-v111-v100-)

J2 = !(001-v0111v0110v111-v110-v100-)

K2 = !(000-v001-v0110v010-v110-v100-)

Минимизированные функции:

y1 = !(!Qt0 !Qt1 v Qt0 Qt1 v !Qt2 )

y2 = !(!Qt0 v !Qt2 v Qt1 )

y56 = !(!Qt0 !Qt1 v !Qt1 !Qt2 v Qt0 Qt1 v Qt1 Qt2 v Qt0 !Qt2 )

J0 = !(!Qt0 Qt1 Qt2 x1 v !Qt0 !Qt1 v !Qt1 !Qt2 v Qt0 Qt1 v Qt0 !Qt2 )

K0 = !(!Qt0 !Qt2 v !Qt0 v Qt1 )

J1 = !(!Qt2 v Qt1 )

K1 = !(!Qt0 !Qt2 v !Qt1 !Qt2 v Qt1 Qt2 v !Qt0 )

J2 = !(!Qt0 !Qt1 Qt2 v Qt0 Qt1 v Qt1 Qt2 v Qt0 !Qt2 )

K2 = !(!Qt0 Qt1 Qt2 !x1 v !Qt0 !Qt1 v !Qt2 )

(тоже что и выше, но в битах):

y1 = !(00--v11--v--0-)

y2 = !(0---v--0-v-1--)

y56 = !(00--v-00-v11--v-11-v1-0-)

J0 = !(0111v00--v-00-v11--v1-0-)

K0 = !(0-0-v0---v-1--)

J1 = !(--0-v-1--)

K1 = !(0-0-v-00-v-11-v0---)

J2 = !(001-v11--v-11-v1-0-)

K2 = !(0110v00--v--0-)

Эффективность минимизации:

1. Эффективность минимизации по количеству входов:

y1 : 4.8

y2 : 10.666666666666666

y56 : 2.8

J0 : 2.0

K0 : 7.0

J1 : 14.0

K1 : 4.0

J2 : 2.6666666666666665

K2 : 3.4285714285714284

2. Эффективность минимизации по количеству блоков:

y1 : 2.2

y2 : 3.0

y56 : 1.4444444444444444

J0 : 1.2222222222222223

K0 : 2.6

J1 : 4.333333333333333

K1 : 1.8571428571428572

J2 : 1.5714285714285714

K2 : 2.2

**Алгоритмы работы:**

Последовательно склеиваются все термы друг с другом формируя множество СДНФ, те что склеивались помечаются.

Затем непомеченные термы составляют таблицу покрытия и в результате появляется МДНФ, которая и есть самая настоящая минимизированная функция.

**ЛИСТИНГ:**

package Minimization;

import java.util.ArrayList;

import java.util.StringTokenizer;

public class Kwaine\_Minimization {

public static double count\_type; //количество неминимизированных входов

public static double count\_mini; //количество минимизированных входов

public static double block\_type; //количество элементов И, ИЛИ до минимизации

public static double block\_mini; //количество элементов И, ИЛИ после минимизации

//формат: "0000v0110v10-1" нАпРиммер

//s - строка для минимизации

public static String minimize(String s) {

StringTokenizer st = new StringTokenizer(s,"v"); //вычисляются все импликанты

ArrayList<String[]> al = new ArrayList<String[]>();

count\_type = 0;

block\_type = 0; //инициализируются "средства" сбора инфы

block\_mini = 0; //об эффективности минимизаций

int count = st.countTokens(); //считается количество импликант

for (int i = 0; i < count; i++) {

String[] ss = {st.nextToken(), "0"}; //создаётся особый массив, позволяющий

al.add(ss); //выделять пройденные и склееные импликанты,

count\_type += ss[0].length();//0 - не склеен, 1 - склеен

}

count\_mini = 0;

block\_type = 2 \* count - 1;

//1.4. Если не находится импликант для склеивания, тогда указатель сдвигается на одно место/элемент поближе к концу. А непомеченная из-за этого импликанта используется потом в МКНФ.

//1.5. Обход списка заканчивается, когда указатель достигает конца списка.

for (int i = 0; i < al.size(); i++) {

//1.2. Из списка берётся импликанта/терм...

char[] first = al.get(i)[0].toCharArray();

//...и пытается склеится со всеми импликантами/термами которые находятся дальше по списку.

for (int j = i + 1; j < al.size(); j++) {

char[] second = al.get(j)[0].toCharArray();

boolean flag = false, toExit = false;

int place = -1;

for (int z = 0; z < second.length & !toExit; z++) {

//если это импликанта одной и той же группы

boolean iff = ((first[z] == '-') & (second[z] == '-'))

| ((first[z] != '-') & (second[z] != '-'));

if (iff) {

//если находится различие в бите между импликантами...

if ((first[z] != second[z]) & !flag) {

//...ставится флаг, что мы возможно нашли для склеивания

place = z; flag = true;

} else if ((first[z] != second[z]) & flag) { //если ещё раз разница

place = -1; toExit = true; //то нифига не склеится, выходим из цикла

}

} else {

toExit = true; //вот тут если это импликанты разных групп. Например 0-0 и -00

}

}

//если всё же у нас флаг говорит, что склеивание возможно

if (!toExit & (place != -1)) {

second[place] = '-'; //на месте разных бит ставим прочерк "-"

al.get(i)[1] = "1"; //и маркируем обе импликанты как склееные

al.get(j)[1] = "1";

String buff = "";

for (int z = 0; z < second.length; z++) {

buff += second[z];

}

String[] ss = {buff , "0"}; //добавляем новую импликанту/терм в список

al.add(ss);

}

}

}

//обход закончился! Ура! СККФ или СДНФ сформированно!

String buff = "";

ArrayList<String> implicants = new ArrayList<String>();

for (int i = 0; i < al.size(); i++) {

if (al.get(i)[1] == "0") { //все непомеченные импликанты,

if (!implicants.contains(al.get(i)[0])) {//и неодинаковые к тому же

implicants.add(al.get(i)[0]); //собираются отдельно!

}

}

}

//обход покрываемости!

for (int i = 0; i < implicants.size() & !al.isEmpty(); i++) {

//берётся импликанта

char[] first = implicants.get(i).toCharArray();

for (int j = 0; j < al.size(); j++) {

//и сравнивается с исходными импликантами побитово

char[] second = al.get(j)[0].toCharArray();

boolean toExit = false;

for (int z = 0; z < second.length & !toExit; z++) {

boolean iff = (first[z] == '-'); //если в бите стоит прочерк

//тогда этот бит покрывает, всё по плану

if (!iff & (first[z] != second[z])) { //если в бите не стояло прочерка

toExit = true; //и к тому же биты разные, значит, увы, импликанта не для этого

} //исходного терма.

} //идём к следующему... терму...

if (!toExit) { //но если всё же покрывает!

al.remove(j); //удаляем исходный терм!

}

}

//тут формируется окончательная строка минимизированной функции.

if (buff != "") {

buff += "v";

block\_mini++;

}

buff += implicants.get(i);

block\_mini++;

}

char[] ww = buff.toCharArray();

for (int i = 0; i < ww.length; i++) {

if ((ww[i] != 'v') & (ww[i] != '-'))

count\_mini++;

}

return buff;

}

//эффективность минимизации по количеству входов

public static double getTermEfficient() {

return count\_type / count\_mini;

}

//эффективность минимизации по количеству элементов/блоков

public static double getBlockEfficient() {

return block\_type / block\_mini;

}

}

**ВЫВОД:**

Я научился реализовывать простейший метод минимизации булевых функций, Квайна-МакКласки, во время работы я встретился с рядом страшных и непонятных вещей, которые усложняли мне разработку. Некоторые сложности программирования имело место быть.